Fundamentals: Difference between Numeric, MA and MVs

This page explains the differecne between the different types of arrays in CDAT and try to explain which context is better suited for each of them.
Unlike Fortran or C, CDAT introduces 3 sperate kinds of arrays.
They all relate to each other and in most cases the differences can be ignored.
All examples in here will be treated with a one dimensional example (for simplicty).

1– Arrays in pure Python
Python does not have an "array" type, this is limitating for scientifc users.
The closest thing to an "array" in Python would be a list.
for example:

```
data=[1.,2.,3.,4.,5.,6.]
```

Unfortunately lists are slow and inefficient for big dataset (especially nested list).

2– The Numeric Module
The basic "array" in CDAT/Python is introduced via the Numeric module.
This module (more info at: http://numeric.scipy.org) offers a wide variety of array manipulation to the user.
It is written in C and therefore offers great speed while applying operators.
It offers any computation, but also things like reshaping, array sorting, matrix manipulation, etc...
In order to convert a Python list to an array, simply use the Numeric.array function:

```
import Numeric
Ndata=Numeric.array(data)
print Ndata.typecode()       # returns 'd' (double)
print Numeric.average(Ndata) # 3.5
```

3– MA Module, missing/masked values

Of course it was a big step to be able to do a fast operation on arrays in Python.
Unfortunately our field requires the introduction of "missing" or "masked" values.
Indeed, observations will often have area where no data exists, also the statisitcs or operator we apply to arrays, might lead to "mathematically wrong" values (division by zero, infinity, etc...)
The Numeric developers were very aware of that fact and developed a module built on top of Numeric called MA (for Masked Arrays).
This module "knows" about masked/missing values and knows how to handle them.
For example let's say we want to mask the 1st and 3rd value in our example.
We would do:

```
import MA
MAdata=MA.array(data,mask=[1,0,1,0,0,0])
print MAdata               # [-- ,2.0 ,-- ,4.0 ,5.0 ,6.0 ,]
print MA.average(MAdata)   # 4.25
```

A lot of functions are available to mask your existing data according to your needs, among others be aware of:
MA.masked_where(condition array,data)
MA.masked_equal(value or array ,data)
MA.masked_greater(value or array,data) , and *masked_less*, *masked_greater_equal*, etc...

4– MV Module, upgrading arrays to "self descriptive variables"

Of course it is nice to be able to perform operation on arrays but it is nicer to know what this array means, things like:

- what does it represent ?
- what is each dimension representing ?
- history ?
- comments ?

That's where the concept of " masked variable" (or MV) comes in.

The MV (maksed variable) module has been built on top of the MA module and operates on "variables".

A variable is essentially a MaskedArray with a name and a description of each dimension (axis) associted with it.
Additional information can be added to it.

To create an MV you will use the MV.array function

```
import MV
MVdata=MV.array(data,mask=[1,0,1,0,0,0])
print MVdata              # [-- ,2.0 ,-- ,4.0 ,5.0 ,6.0 ,]
print MV.average(MVdata)  # 4.25
MVdata.info()
*** Description of Slab variable_6 ***
id: variable_6
shape: (6,)
filename:
missing_value: None
comments:
grid_name: N/A
grid_type: N/A
time_statistic:
long_name:
units:
No grid present.
** Dimension 1 **
   id: axis_0
   Length: 6
   First:  0.0
   Last:   5.0
   Python id:  0x40fd002c
*** End of description for variable_6 ***
```

Note how the last command *MVdata.info()* returns all sorts of information about the variable, its name and the axes associated with each dimension.

In essence the MV module operates the same way as the MA module but it will try to preserve the metadata as much as possible.

5– Conclusion and Tips on how/when to use each module

In short we learned that Numeirc was the C based efficent way to deal with big array operations in Python.
That MA was built on top of it to remedy the need to deal with "masked values".
And that data information required that yet another module was built on top of it to allow data information to

be added to the array, the MV module.

Of course each new layer adds functionality but takes away some speed as more query need to be made on the data.

There's basically 2 ways to deal with these problems:

1– Only use MV
  MV has all the features of MA and Numeric, if your code is still fast enough don't bother

2– If speed is an issue
 save the metada you'll need (see other exmaples in order to learn how to do so)
 work at the MA level or Numeric level if you don't have to deal with masked values
 at the end of your computations put the metadata back on and display/save your results.